# 嵌入式 Linux 与物联网软件开发
# ——C 语言内核深度解析

人 民 邮 电 出 版 社
北 京

# 前言

PREFACE

C 语言是嵌入式 Linux 领域的主要开发语言。对于学习嵌入式、单片机、Linux 驱动开发等技术来说，C 语言是必须要过的一关。C 语言学习的特点是入门容易、深入理解难、精通更是难上加难。很多用 C 语言写了多年单片机程序的老工程师转入嵌入式 Linux 领域后，都会觉得很难，甚至惊叹“为什么同样是 C 语言代码，我完全看不懂？”更不用说初学者了，大多数人都会有一种“很难精进、很难掌握”的感觉。

本书就是为了解决这个问题。朱有鹏老师在由嵌入式软件开发人员转为职业培训讲师后，试图找到一种方式能够将研发实践中的技能和技巧传授给学生，而不仅仅是冰冷晦涩的语法和知识点。没错，我们认为 C 语言既是一门技艺，也是一种能力，就好像开车、踢足球、厨艺等一样，不只是要“知道怎么回事儿”，还要“玩儿得好”才行。

本书的原型思想和内容，发源于朱有鹏老师早些年的研发和学习经历，发展于后来数年的线下培训授课经历，并最终成熟于视频课程《4.C 语言高级专题》(隶属于《朱有鹏老师嵌入式 Linux 核心课程》系列视频课程的第 4 部分)。该套视频课程于 2015 年 10 月录制完成，并在不到的一年时间内，已被上千人观看学习，创下了全好评的好成绩。

本书正是基于这套视频课程的课件整理而来，参与各章节整理和编写的都是学习了视频课程的学生，最终由朱有鹏老师和张先凤老师检验并完善成书。这些参与编写的同学有的已经工作数年、有的则尚未走出大学校园。选择他们合作创作本书，就是为了告诉读者：做技术并不要求你天赋异禀，只需要你感兴趣、愿意去探索和练习，你也可以成功。

本书的另一大特色是，专门针对嵌入式 Linux 开发方向而设计。这并不是一句空话，本书的很多内容，如位操作、container_of 宏、内核链表、变参等，都是嵌入式 Linux 开发中重要的技能，而在一般的 C 语言书中并无过多介绍。

最后，本书并不是一本零基础系统学习 C 语言的书，而是一本定位为技能提升型的专著。如果你已经学过或者正在使用 C 语言，但苦于无法精进，或者在学习嵌入式 Linux 软件开发中遇到困难，那么试试这本书吧，一定会为你带来收获。

# 目录

CONTENTS

**第 1 章　C 语言与内存** …… 1

1.1　引言 …… 1

1.2　计算机程序运行的目的 …… 1

1.2.1　什么是程序 …… 1

1.2.2　计算机运行程序的目的 …… 1

1.2.3　静态内存 SRAM 和动态内存 DRAM …… 2

1.2.4　冯・诺伊曼结构和哈佛结构 …… 3

1.2.5　总结：程序运行为什么需要内存呢 …… 4

1.2.6　深入思考：如何管理内存（无 OS 时，有 OS 时） …… 4

1.3　位、字节、半字、字的概念和内存位宽 …… 5

1.3.1　深入了解内存（硬件和逻辑两个角度） …… 5

1.3.2　内存的逻辑抽象图（内存的编程模型） …… 6

1.3.3　位和字节 …… 7

1.3.4　字和半字 …… 7

1.3.5　内存位宽（硬件和逻辑两个角度） …… 7

1.4　内存编址和寻址、内存对齐 …… 8

1.4.1　内存编址方法 …… 8

1.4.2　关键：内存编址是以字节为单位 …… 8

1.4.3　内存和数据类型的关系 …… 9

1.4.4　内存对齐 …… 10

1.5　C语言如何操作内存 …… 10

1.5.1　C 语言对内存地址的封装 …… 10

1.5.2　用指针来间接访问内存 …… 11

1.5.3　指针类型的含义 …… 12

1.5.4　用数组来管理内存 …… 12

1.6　内存管理之结构体 …… 14

1.6.1　数据结构这门学问的意义 …… 14

1.6.2　最简单的数据结构：数组 …… 14

1.6.3 数组的优缺点 …… 14
1.6.4 结构体隆重登场 …… 14
1.6.5 题外话：结构体内嵌指针实现面向对象 …… 15
1.7 内存管理之栈（stack） …… 15
1.7.1 什么是栈 …… 15
1.7.2 栈管理内存的特点（小内存、自动化） …… 16
1.7.3 栈的应用举例：局部变量和函数调用 …… 16
1.7.4 栈的约束（预定栈大小不灵活，怕溢出） …… 17
1.8 内存管理之堆 …… 18
1.8.1 什么是堆 …… 18
1.8.2 堆管理内存的特点（大块内存、手工分配 / 使用 / 释放） …… 18
1.8.3 C 语言操作堆内存的接口（malloc/free） …… 18
1.8.4 堆的优势和劣势（管理大块内存、灵活、容易内存泄漏） …… 19
1.8.5 静态存储区 …… 19
课后题 …… 19

**第 2 章 C 语言位操作 …… 21**

2.1 引言 …… 21
2.2 常用位操作符 …… 21
2.2.1 位与（&） …… 21
2.2.2 位或（|） …… 22
2.2.3 位取反（~） …… 23
2.2.4 位异或（^） …… 24
2.2.5 左移位（<<） …… 25
2.2.6 右移位（>>） …… 26
2.3 位操作与寄存器 …… 26
2.3.1 寄存器的操作 …… 26
2.3.2 寄存器特定位清零用 & …… 27
2.3.3 寄存器特定位置 1 用 | …… 27
2.3.4 寄存器特定位取反用 ~ …… 27
2.4 位运算构建特定二进制数 …… 28
2.4.1 使用移位获取特定位为 1 的二进制数 …… 28
2.4.2 结合位取反获取特定位为 0 的二进制数 …… 29
2.4.3 总结 …… 29
2.5 位运算实战演练1 …… 30
2.5.1 给定整型数 a，设置 a 的 bit3，保证其他位不变 …… 30
2.5.2 给定整型数 a，设置 a 的 bit3~bit7，保持其他位不变 …… 30
2.5.3 给定整型数 a，清除 a 的 bit15，保证其他位不变 …… 30

2.5.4 给定整型数 a，清除 a 的 bit15~bit23，保持其他位不变 …… 31
2.5.5 给定整型数 a，取出 a 的 bit3~bit8 …… 31
2.5.6 用 C 语言给寄存器 a 的 bit7~bit17 赋值 937（其余位不受影响）…… 31
2.6 位运算实战演练2 …… 32
2.6.1 用 C 语言将寄存器 a 的 bit7~bit17 中的值加 17（其余位不受影响）…… 32
2.6.2 用 C 语言给寄存器 a 的 bit7~bit17 赋值 937，同时给 bit21~bit25 赋值 17 …… 32
2.7 技术升级：用宏定义来完成位运算 …… 33
2.7.1 直接用宏来置位 …… 33
2.7.2 直接用宏来复位 …… 33
2.7.3 截取变量的部分连续位 …… 33
课后题 …… 34

**第 3 章 指针才是 C 语言的精髓 …… 36**

3.1 引言 …… 36
3.2 指针到底是什么 …… 37
3.2.1 普通变量 …… 37
3.2.2 指针变量 …… 37
3.2.3 变量空间的首字节地址，作为整个空间的地址 …… 38
3.2.4 指针变量的类型作用 …… 38
3.2.5 为什么需要指针 …… 39
3.2.6 高级语言如 Java、C# 的指针到哪里去了 …… 39
3.2.7 指针使用之三部曲 …… 40
3.3 理解指针符号 …… 40
3.3.1 星号 * 的理解 …… 40
3.3.2 取地址符 & 的理解 …… 41
3.3.3 指针变量的初始化和指针变量赋值之间的区别 …… 41
3.3.4 左值与右值 …… 41
3.3.5 定义指针后，需要关心的一些内容 …… 42
3.4 野指针与段错误问题 …… 43
3.4.1 什么是野指针 …… 43
3.4.2 野指针可能引发的危害 …… 44
3.4.3 野指针产生的原因 …… 44
3.4.4 如何避免野指针 …… 45
3.4.5 NULL 到底是什么 …… 45
3.4.6 段错误产生的原因汇总 …… 46
3.5 const关键字与指针 …… 46
3.5.1 什么是 const …… 46
3.5.2 const 对于普通变量的修饰 …… 46

3.5.3 const 修饰指针的三种形式 …… 46
3.5.4 const 的变量真的不能改吗 …… 47
3.5.5 为什么要用 const …… 48
3.5.6 有关变量和常量的探讨 …… 48
3.6 深入学习数组 …… 49
3.6.1 为什么需要数组 …… 49
3.6.2 从编译器角度理解数组 …… 49
3.6.3 从内存角度理解数组 …… 49
3.6.4 一位数组中几个关键符号的理解 …… 50
3.7 指针与数组的天生“姻缘” …… 50
3.7.1 如何使用指针访问数组 …… 50
3.7.2 从内存角度理解指针访问数组的实质 …… 51
3.7.3 指针与数组类型的匹配问题 …… 51
3.7.4 总结：指针类型决定了指针如何参与运算 …… 51
3.8 指针类型与强制类型转换 …… 52
3.8.1 变量数据类型的作用 …… 52
3.8.2 数据的存入和读取 …… 53
3.8.3 普通变量的强制转换 …… 53
3.8.4 指针变量数据类型的含义 …… 56
3.8.5 指针变量数据类型的强制转换 …… 56
3.9 指针、数组与sizeof运算符 …… 57
3.9.1 char str[]="hello"; sizeof(str)，sizeof(str[0])，strlen(str) …… 58
3.9.2 char str[]="hello"; char *p=str; sizeof(*p) …… 58
3.9.3 int b[100]; sizeof(b) …… 58
3.9.4 数组的传参 …… 58
3.9.5 #define 和 typedef 的区别 …… 59
3.10 指针与函数传参 …… 60
3.10.1 普通传参 …… 60
3.10.2 传递地址（指针）…… 61
3.10.3 传递数组 …… 61
3.10.4 传递结构体 …… 61
3.10.5 传递普通值和传递地址的异同，以及传递地址（指针）应该遵循的原则 …… 62
3.11 输入型参数与输出型参数 …… 62
3.11.1 函数为什么需要传参和返回值 …… 62
3.11.2 函数传参中为什么使用 const 指针 …… 64
3.11.3 总结 …… 65
课后题 …… 65

第 4 章　C 语言复杂表达式与指针高级应用 …… 67
4.1　引言 …… 67
4.2　指针数组与数组指针 …… 67
4.2.1　简单理解指针数组与数组指针 …… 67
4.2.2　分析指针数组与数组指针的表达式 …… 68
4.3　函数指针与typedef …… 69
4.3.1　函数指针的实质（还是指针变量） …… 70
4.3.2　函数指针的书写和分析方法 …… 70
4.3.3　typedef 关键字的用法 …… 72
4.4　函数指针实战1——用函数指针调用执行函数 …… 73
4.5　函数指针实战2——结构体内嵌函数指针实现分层 …… 75
4.6　再论typedef …… 78
4.6.1　轻松理解和应用 typedef …… 78
4.6.2　typedef 与 #define 宏的区别 …… 80
4.6.3　typedef 与 struct …… 81
4.6.4　typedef 与 const …… 81
4.6.5　使用 typedef 的重要意义 …… 82
4.6.6　二重指针 …… 82
4.7　二维数组 …… 83
4.7.1　二维数组的内存映像 …… 83
4.7.2　识别第一维和第二维 …… 83
4.7.3　数组名代表数组首元素的地址 …… 84
4.7.4　指针访问二维数组的两种方式 …… 84
4.7.5　总结 …… 85
课后题 …… 85

第 5 章　数组 & 字符串 & 结构体 & 共用体 & 枚举 …… 88
5.1　引言 …… 88
5.2　程序中的内存从哪里来 …… 88
5.2.1　三种内存来源：栈（stack）、堆（heap）、数据区（.data） …… 89
5.2.2　栈内存特点详解 …… 89
5.3　堆 …… 90
5.3.1　堆内存特点详解 …… 90
5.3.2　使用堆内存注意事项 …… 91
5.3.3　malloc 的一些细节表现 …… 92
5.4　内存中的各个段 …… 92
5.4.1　代码段、数据段、bss 段 …… 92
5.4.2　特殊数据会被放到代码段 …… 92

5.4.3　未初始化或显式初始化为 0 的全局变量放在 bss 段 …… 93
5.4.4　内存管理方式的总结 …… 93
5.5　C语言的字符串类型 …… 93
5.5.1　C 语言使用指针来管理字符串 …… 93
5.5.2　C 语言中字符串的本质：指向字符串的存放空间的指针 …… 94
5.5.3　指向字符串的指针变量空间和字符串存放的空间是分开的 …… 94
5.5.4　存储多个字符的两种方式——字符串和字符数组 …… 94
5.6　字符串和字符数组的细节 …… 95
5.6.1　字符数组的初始化、sizeof 以及 strlen …… 95
5.6.2　字符串的初始化与 sizeof、strlen …… 96
5.6.3　字符数组与字符串的本质差异 …… 96
5.7　结构体概述 …… 96
5.7.1　结构体使用时先定义结构体类型，再用类型定义变量 …… 96
5.7.2　从数组到结构体的进步之处 …… 97
5.7.3　结构体变量中的元素如何访问 …… 97
5.8　结构体的对齐访问 …… 98
5.8.1　结构体对齐访问实例 …… 98
5.8.2　结构体为何要对齐访问 …… 99
5.8.3　结构体对齐的规则和运算 …… 99
5.8.4　手动对齐 …… 100
5.8.5　GCC 推荐的对齐指令：_attribute_((packed)) 和 _attribute_((aligned(n))) …… 102
5.9　offsetof宏与container_of宏 …… 103
5.9.1　由结构体指针进而访问各元素的原理 …… 103
5.9.2　offsetof 宏 …… 104
5.9.3　container_of 宏 …… 104
5.9.4　学习指南和要求 …… 105
5.10　共用体（union） …… 105
5.10.1　共用体的类型声明、变量定义和使用 …… 105
5.10.2　共用体和结构体的区别 …… 106
5.10.3　共用体的主要用途 …… 106
5.11　大小端模式 …… 107
5.11.1　什么是大小端模式 …… 107
5.11.2　用 union 来测试机器的大小端模式 …… 108
5.11.3　用指针方式来测试机器的大小端 …… 109
5.11.4　通信系统中的大小端（数组的大小端） …… 109
5.12　枚举enum …… 110
5.12.1　枚举的作用是什么 …… 110
5.12.2　C 语言为何需要枚举 …… 111

5.12.3 宏定义和枚举的区别 ······ 111
5.12.4 枚举的定义形式 ······ 111
课后题 ······ 113
**第 6 章 C 语言的预处理、函数和函数库 ······ 116**
6.1 引言 ······ 116
6.2 C语言为什么需要编译链接 ······ 116
6.2.1 编译链接的流程 ······ 116
6.2.2 编译链接中各种文件扩展名的含义 ······ 118
6.3 预处理详解 ······ 119
6.3.1 C 语言预处理的意义 ······ 119
6.3.2 预处理涉及的内容 ······ 119
6.3.3 使用 GCC 进行编译和链接的过程 ······ 119
6.4 常见的预处理详解 ······ 120
6.4.1 文件包含 ······ 120
6.4.2 注释 ······ 122
6.4.3 宏定义 ······ 123
6.4.4 条件编译 ······ 126
6.5 函数的本质 ······ 129
6.5.1 C 语言为什么会有函数 ······ 129
6.5.2 函数书写的一般原则 ······ 129
6.5.3 函数是动词、变量是名词（面向对象中分别叫方法和成员变量） ······ 129
6.5.4 函数的实质是数据处理器 ······ 130
6.6 函数的基本使用 ······ 130
6.6.1 函数三要素：定义、声明、调用 ······ 130
6.6.2 函数原型和作用 ······ 131
6.7 递归函数 ······ 131
6.7.1 函数的调用机制 ······ 131
6.7.2 递归函数 ······ 132
6.7.3 使用递归的原则：收敛性、栈溢出 ······ 134
6.7.4 递归与循环的区别 ······ 135
6.8 库函数 ······ 135
6.8.1 什么是函数库 ······ 135
6.8.2 函数库的由来 ······ 135
6.8.3 函数库的提供形式：静态链接库与动态链接库 ······ 135
6.8.4 库函数的使用 ······ 136
6.9 常见的库函数之字符串函数 ······ 138
6.9.1 什么是字符串 ······ 138

6.9.2 字符串处理函数 ········· 138
6.9.3 man 手册的引入 ········· 138
6.9.4 man 手册的使用 ········· 138
6.9.5 常用的字符串处理函数 ········· 139
6.10 常见的库函数之数学库函数 ········· 140
6.10.1 数学库函数 ········· 140
6.10.2 计算开平方 ········· 140
6.10.3 链接时加 -lm ········· 141
6.11 制作静态链接库并使用 ········· 141
6.12 制作动态链接库并使用 ········· 143
课后题 ········· 144

**第 7 章 存储类 & 作用域 & 生命周期 & 链接属性 ········· 147**

7.1 引言 ········· 147
7.2 概念解析 ········· 147
7.2.1 存储类 ········· 147
7.2.2 作用域 ········· 148
7.2.3 生命周期 ········· 148
7.2.4 链接属性 ········· 148
7.3 Linux下C程序的内存映像 ········· 149
7.3.1 代码段、rodata 段（只读数据段）········· 149
7.3.2 数据段、bss 段 ········· 149
7.3.3 堆 ········· 149
7.3.4 文件映射区 ········· 149
7.3.5 栈 ········· 150
7.3.6 内核映射区 ········· 150
7.3.7 操作系统下和裸机下 C 程序加载执行的差异 ········· 150
7.4 存储类相关的关键字1 ········· 150
7.4.1 auto ········· 150
7.4.2 static ········· 151
7.4.3 register ········· 151
7.5 存储类相关的关键字2 ········· 152
7.5.1 extern ········· 152
7.5.2 volatile ········· 153
7.5.3 restrict ········· 154
7.5.4 typedef ········· 154
7.6 作用域详解 ········· 154
7.6.1 局部变量的代码块作用域 ········· 154

7.6.2 函数名和全局变量的文件作用域 …… 155

7.7 变量的生命周期 …… 155

7.7.1 研究变量生命周期的意义 …… 155

7.7.2 栈变量的生命周期 …… 156

7.7.3 堆变量的生命周期 …… 156

7.7.4 数据段、bss 段变量的生命周期 …… 156

7.7.5 代码段、只读段的生命周期 …… 156

7.8 链接属性 …… 156

7.8.1 C 语言程序的组织架构：多个 C 文件 + 多个 h 文件 …… 156

7.8.2 编译以文件为单位、链接以工程为单位 …… 157

7.8.3 三种链接属性：外连接、内链接、无链接 …… 157

7.8.4 函数和全局变量的命名冲突问题 …… 157

7.8.5 static 的第二种用法：修饰全局变量和函数 …… 158

课后题 …… 158

**第 8 章 C 语言关键细节讨论 …… 159**

8.1 引言 …… 159

8.2 操作系统概述 …… 159

8.2.1 什么是操作系统 …… 159

8.2.2 C 库函数 …… 160

8.2.3 操作系统的重大意义 …… 161

8.3 main函数返回值 …… 162

8.3.1 普通函数的返回值 …… 162

8.3.2 main 函数的返回值 …… 163

8.3.3 谁调用了 main 函数 …… 164

8.4 argc、argv与main函数的传参 …… 164

8.5 void类型的本质 …… 165

8.6 C语言中的NULL …… 166

8.6.1 NULL 的定义 …… 166

8.6.2 '\0' 、'0' 、0 和 NULL 的区别 …… 167

8.7 运算中的临时匿名变量 …… 168

8.7.1 C 语言和汇编语言的区别 …… 168

8.7.2 强制类型转换 …… 168

8.7.3 使用临时变量来理解不同数据类型之间的运算 …… 169

8.8 顺序结构 …… 169

8.8.1 C 语言中的结构 …… 169

8.8.2 编译过程中的顺序结构 …… 169

8.8.3 思考：为什么本质都是顺序结构 …… 170

8.9 程序调试 ······ 170
8.9.1 程序调试手段 ······ 170
8.9.2 调试 (DEBUG) 版本和发行 (RELEASE) 版本的区别 ······ 171
8.9.3 debug 宏的使用方法 ······ 171
课后题 ······ 172

**第 9 章 链表 & 状态机 & 多线程 ······ 173**

9.1 引言 ······ 173
9.2 链表的引入 ······ 173
9.2.1 数组的缺陷 ······ 173
9.2.2 链表是什么样子的 ······ 174
9.2.3 链表的作用是什么 ······ 174
9.3 单链表的实现之构建第一个节点 ······ 175
9.3.1 不实用却意义重大的简单链表 ······ 175
9.3.2 从简单的链表开始 ······ 175
9.3.3 单链表的节点构成 ······ 175
9.3.4 使用堆内存创建一个节点 ······ 176
9.3.5 链表的头指针 ······ 177
9.3.6 构建第一个简单的链表 ······ 177
9.4 单链表的实现之从尾部插入节点 ······ 177
9.4.1 从尾部插入节点 ······ 177
9.4.2 构建第一个简单的链表 ······ 178
9.4.3 什么是头节点 ······ 179
9.5 单链表的算法之从头部插入节点 ······ 180
9.5.1 链表头部插入思路解析 ······ 180
9.5.2 箭头非指向 ······ 181
9.6 单链表的算法之遍历节点 ······ 181
9.6.1 什么是遍历 ······ 181
9.6.2 如何遍历单链表 ······ 181
9.6.3 代码分析 ······ 182
9.7 单链表的算法之删除节点 ······ 182
9.7.1 为什么要删除节点 ······ 182
9.7.2 注意堆内存的释放 ······ 183
9.7.3 设计一个删除节点算法 ······ 184
9.8 单链表的算法之逆序 ······ 184
9.8.1 什么是链表的逆序 ······ 184
9.8.2 单链表的逆序算法分析 ······ 185
9.8.3 编程实现逆序算法 ······ 185

9.8.4 数据结构与算法的关系 …… 186
9.9 双链表的引入和基本实现 …… 187
9.9.1 单链表的优缺点 …… 187
9.9.2 双链表的结构 …… 187
9.10 双链表的算法之插入节点 …… 188
9.10.1 尾部插入 …… 188
9.10.2 头部插入 …… 189
9.11 双链表的算法之遍历 …… 190
9.11.1 正向遍历 …… 190
9.11.2 逆向遍历 …… 190
9.12 双链表的算法之删除节点 …… 191
9.13 Linux内核链表 …… 192
9.13.1 前述链表数据区域的局限性 …… 193
9.13.2 解决思路：数据区的结构体的封装由用户实现，通用部分通过调用函数实现 …… 193
9.13.3 内核链表的设计思路 …… 193
9.13.4 list.h 文件简介 …… 194
9.14 内核链表的基本算法和使用简介 …… 196
9.14.1 内核链表的常用操作 …… 196
9.14.2 内核链表的使用实践 …… 197
9.15 什么是状态机 …… 198
9.15.1 有限状态机 …… 199
9.15.2 两种状态机：Moore 型和 Mealy 型 …… 199
9.15.3 状态机的主要用途 …… 200
9.15.4 状态机解决了什么问题 …… 200
9.16 用C语言实现简单的状态机 …… 201
9.16.1 题目：开锁状态机 …… 201
9.16.2 题目分析 …… 201
9.17 多线程简介 …… 203
9.17.1 操作系统下的并行执行机制 …… 203
9.17.2 进程和线程的区别和联系 …… 204
9.17.3 多线程的优势 …… 204
课后题 …… 204

**第 10 章 程序员和编译器的暧昧 …… 206**

10.1 引言 …… 206
10.2 编程工作的演进史 …… 206
10.2.1 CPU 与二进制 …… 206
10.2.2 编程语言的革命 …… 207

10.3 程序员、编译器和CPU之间的三角恋 …… 207
10.3.1 程序员与 CPU 的之间的“翻译”——编译器 …… 207
10.3.2 高级语言与低级语言的差别 …… 208
10.4 像编译器一样思考吧——理论篇 …… 208
10.4.1 编译器的结构 …… 208
10.4.2 语法是什么？语法就是编译器的习性 …… 208
10.5 像编译器一样思考吧——实战篇 …… 209
10.5.1 充分地利用语法规则，写出简洁、高效的代码 …… 209
10.5.2 复杂表达式理解 …… 209
课后题 …… 214

**附录 答案 …… 215**

第1章 课后题答案 …… 215
第2章 课后题答案 …… 217
第3章 课后题答案 …… 218
第4章 课后题答案 …… 220
第5章 课后题答案 …… 222
第6章 课后题答案 …… 226
第7章 课后题答案 …… 228
第8章 课后题答案 …… 230
第9章 课后题答案 …… 231
第10章 课后题答案 …… 236